**ECE 414 – Embedded Systems**

**Report for Lab 02 – PIC 32 GPIO and C Programming**

GuoYuan Li (Scribe)

Harry Zhu

Sep 1, 2020

**1. Introduction**

The objective of this lab is to increase the familiarity with C programming, especially writing header files, using a modular structure, which divides the task into smaller modules for readability and modularity. More essentially, this lab provides an opportunity to learn more about SFRs that configure input and output ports. In addition, we explored further about the hardware implementation on breadboard with both input and output connections. Finally, we learned to write test codes to test the correctness of both software and the hardware.

**2. Requirements**

Two essential modules porta\_in and portb\_out are created in the lab to configure the input and output. Specifically, detailed requirements are listed below:

1. Module portb\_out shall implement an initialization function named portb\_out\_init()that will configure PORTB as an output port that can write all port bits that are available on the output pins of the ’128B (specifically, outputs RB0-RB5, RB7-RB11, and RB13-RB15).
2. Module portb\_out shall implement a function named portb\_out\_write(uint16\_t a) that accepts a 16-bit unsigned integer as input and writes the 14 least significant bits of that value to output pins {RB15:13,RB11:RB7,RB5:RB0}. In other words, this function writes a 14-bit binary value while “skipping over” the PORTB pins that do not have external connections to the ‘128B.
3. Module porta\_in shall implement an initialization function named porta\_in\_init()that will configure PORTA as an input port using internal pullup resistors and to read switch inputs on all available PORTA pins on the ‘128B, specifically pins {RA4:RA0}.
4. Module porta\_in shall implement a function named porta\_in\_read()that returns an 8-bit unsigned integer that contains the values {3’b0, RA4:RA0}. Note that the internal pullups mean we can connect a switch to each input pin that is connected to ground; this will be read as a “1” when the switch is open, and as a “0” when the switch is closed.
5. The port modules shall be tested using a circuit similar to Figure 1 but expanded to test all PORTA inputs and PORTB outputs. The schematic diagram for this circuit shall be drawn using KiCad.
6. The port modules shall be tested using a main program should include initializes the port modules and then enters an infinite while (1) loop that performs the following functions:
   1. When input RA4 is a logic low: turn on the output LED corresponding to the binary number encoded on switches RA3:RA0 while turning off all other outputs.
   2. When input RA4 is a logic high: turn off the output LED corresponding to the binary number encoded on switches RA3:RA0 while turning on all other outputs.

**3. Design Description**

**3.1 Hardware Description**

In the lab, we modified the original schematic by connecting the additional 6 LEDs to ports RB 9~15(excluding PORTB 6 and 12) and 360 ohm resistors in series. The DIP switch connected to ports RA 0~4 was configured to be an internal pullup such that it is high when it is open and low when it is connected. The KiCad Schematic is included below in figure 1.

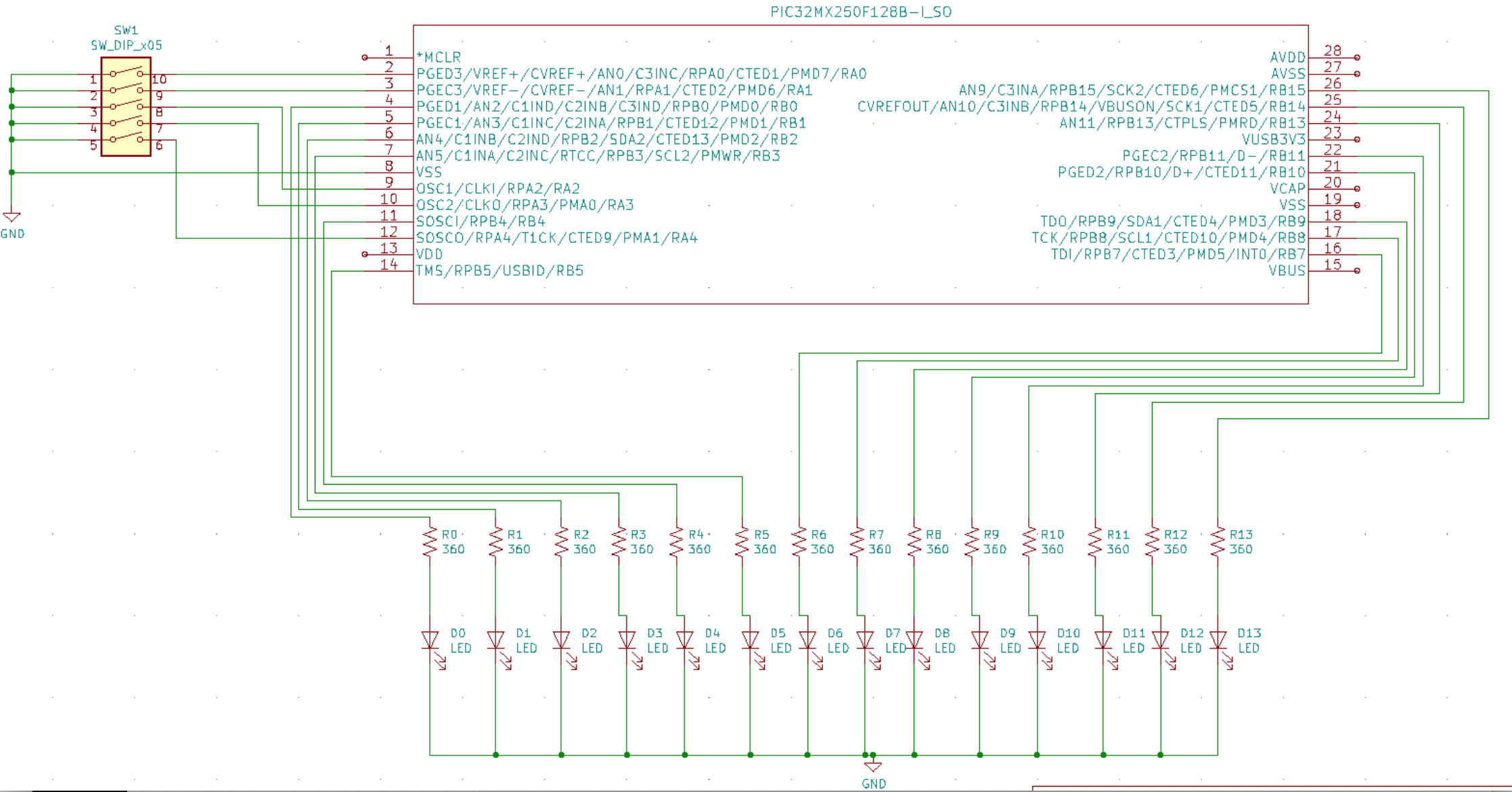


Figure 1 - Modified I/O Test Circuits

**3.2 Software Description**

To fulfill the requirements, we divided the task into 3 main modules: porta\_in, portb\_out, and main. Porta\_in provides input functions that initializes port RA0:4 as input pins with internal pullup and reads the 5-bit information sent from the switch. Portb\_out provides output functions that initializes port RB 0-15, except RB6 and 12, as output pins and configure the output bits’ arrangement using these 14 pins. These two files have both header files, where functions are declared, and source files, where functions are implemented. Finally, the main file includes these header files and source files of porta\_in and portb\_out to read from the switch and write to the LEDs in while(1) loop. There is an additional function in this main.c file that converts the input to desired output called output(uint8\_t in), which accepts an 8-bit binary input from porta\_in and decodes it into the desired output with only 1 bit high at a time. The detailed code implementation is included in section 6 below.

**4. Test Report**

**4.1 Unit Test**

Since both porta\_in and portb\_out modules configure the input/output ports, there are no software tests that can be performed to test their functionality. Therefore, they are integrated with the main program and implemented on hardware to test their functionality, which is described in 4.2 Acceptance Test.

**4.2 Acceptance Test**

This section describes the tests to be performed to test the correctness of the system. Requirements verified for each test will be shown in the traceability matrix in figure 2.

**T1. Hardware Implementation is coherent with the Schematic**

* Pass: Actual hardware implementation resembles the KiCad Schematic
* Fail: Hardware not resemble the KiCad Schematic

**T2.** **Switch is at 0x1F(not connected**)

* Pass: No LED lighted up
* Fail: Some/all LED lighted up

**T3**. **Switch is at 0x00(connected)**

* Pass: No LED lighted up
* Fail: Some/All LED lighted up

**T4. Switch is at 0x10**

* Pass: All 14 LEDs lighted up
* Fail: Not all 14 LEDs lighted up

**T5. Switch is at 0x0F**

* Pass: All 14 LEDs lighted up
* Fail: Not all 14 LEDs lighted up

**T6. Switch is at 0x03**

* Pass: Only the third LED lighted up
* Failed: LEDs other than the third one lightened up or the third one did not light up

**T7. Switch is 0x13**

* Pass: Only the third LED didn’t light up while the others all lighted up
* Fail: any other scenarios other than the pass scenario described above

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Requirements | | | | | | | |
| Tests | 2.1 | 2.2 | 2.3 | 2.4 | 2.5 | 2.6.a | 2.6.b | result |
| T1. Harry |  |  |  |  | x |  |  | pass |
| T2. GuoYuan | x | x | x | x | x |  | x | pass |
| T3. GuoYuan | x | x | x | x | x | x |  | pass |
| T4. Harry | x | x | x | x | x |  | x | pass |
| T5. GuoYuan | x | x | x | x | x | x |  | pass |
| T6. Harry | x | x | x | x | x | x |  | pass |
| T7. GuoYuan | x | x | x | x | x |  | x | pass |

Figure 2. Traceability Matrix

**5. Conclusion**

The lab went successfully as all the tests were passed and all the requirements were satisfied. We’ve learned how to configure input/output ports using SFRs and explored further in writing C programs by learning the module structure and functions. We spent approximately 3 hours in this lab.

**6. Software Implementation**

**6.1 porta\_in.h**

#ifndef PORTA\_IN\_H

#define PORTA\_IN\_H

#include <inttypes.h>

extern void porta\_in\_init();

extern uint8\_t porta\_in\_read();

#endif

**6.2 portb\_out.h**

#ifndef PORTB\_OUT\_H

#define PORTB\_OUT\_H

#include <inttypes.h>

extern void portb\_out\_init();

extern void portb\_out\_write(uint16\_t val);

#endif

**6.3 porta\_in.c**

#include <xc.h>

#include <inttypes.h>

#include "porta\_in.h"

void porta\_in\_init() {

// low-level code to

// initialize port

ANSELA = 0;

TRISA = 0x1F;

CNPUA = 0x1F;

}

uint8\_t porta\_in\_read() {

// low-level code to

// read port

return (PORTA & 0x1F); //0x1F

}

**6.4 portb\_out.c**

#include <xc.h>

#include <inttypes.h>

#include "portb\_out.h"

void portb\_out\_init() {

// low-level code to

// initialize port

ANSELB = 0;

TRISB = 0;

}

void portb\_out\_write(uint16\_t val)

{

// low-level code to

// write port

uint16\_t temp1 = val & 0x003F;

uint16\_t temp2 = (val & 0x07C0) << 1;

uint16\_t temp3 = (val & 0x3800) << 2;

LATB = temp1 | temp2 | temp3;

}

**6.5 main.c**

#pragma config FNOSC = FRCPLL, POSCMOD = OFF

#pragma config FPLLIDIV = DIV\_2, FPLLMUL = MUL\_20

#pragma config FPBDIV = DIV\_1, FPLLODIV = DIV\_2

#pragma config FWDTEN = OFF, JTAGEN = OFF, FSOSCEN = OFF

#include <xc.h>

#include <inttypes.h>

#include "porta\_in.h"

#include "portb\_out.h"

/\*\*

\* This output function accepts a 8-bit input

\* and output 16-bit output representing the

\* binary value encoded on the switch

\*\*/

uint16\_t output(uint8\_t in){

uint16\_t out;

out = 0;

switch(in){

case 0x01:

out = 0x0001;

break;

case 2:

out = 0x0002;

break;

case 3:

out = 0x0004;

break;

case 4:

out = 0x0008;

break;

case 5:

out = 0x0010;

break;

case 6:

out = 0x0020;

break;

case 7:

out = 0x0040;

break;

case 8:

out = 0x0080;

break;

case 9:

out = 0x0100;

break;

case 10:

out = 0x0200;

break;

case 11:

out = 0x0400;

break;

case 12:

out = 0x0800;

break;

case 13:

out = 0x1000;

break;

case 14:

out = 0x2000;

break;

default:

out = 0;

break;

}

return out;

}

main() {

uint8\_t in;

uint16\_t out;

porta\_in\_init();

portb\_out\_init();

in = 0;

out = 0;

while (1) {

in = porta\_in\_read();

if(!(in & 0x10)){

out = output(in);

}else{

out = ~output(in);

}

portb\_out\_write(out);

}

**}**